DS-015 UAVCAN Drone Standard

Revision: 0.8.0
Revision date: Oct 26, 2020

Abstract
This document is the formal version of the UAVCAN Drone Standard developed by the Dronecode Foundation UAVCAN Drone Special Interest Group.

Official Drone Profile of

UAVCAN

Table of contents

Table of contents 2

Document revisions 4

Contact and public developer call 4

Trademark guideline 5

License and disclaimer 5

Terminology and acronyms 6

Introduction 7

Protocol layers and standards 9

- Application layer 9
- Presentation layer 9
- Transport layer 9
- Physical layer 10

Application layer 11

- UAVCAN standard application layer 11
  - Conventions 11
  - Heartbeat 11
  - Introspection 11
  - Register API 11
  - Plug-and-play 12
  - Bootloader 12
- Drone application layer 13
  - Domain-specific services 13
  - Implementation guidelines 13
  - System integration guidelines 14
  - Future capabilities 15

Physical layer 16

- General recommendations 16

Integrated power supply network 16

Power input 16

Power output 17

UAVCAN/CAN recommendations 17

Physical connector specification 17

UAVCAN/CAN Micro connector 18

Future capabilities 19

CAN bus physical layer parameters 19

Classic CAN 19

CAN FD 20

Profiles **21**

CAN bus topologies 21

Non-redundant daisy-chain topology 21

Redundant daisy-chain topology 21

Star topology 22

Redundant transports 22

Quadrotor 23

Quad VTOL 23

Conformance **24**

Document revisions

Revision	Editor	Reviewer	Comments
0.0.1	Nuno Marques	Lorenz Meier	Initial specification
0.2.0	Nuno Marques	Lorenz Meier	Detailed specification for initial message set.
0.3.0	Nuno Marques	Lorenz Meier	Reduced message set with alpha level specification
0.4.0	Nuno Marques	Lorenz Meier	Define port ID ranges and mechanisms
0.5.0	Nuno Marques	Lorenz Meier	Add physical layer recommendations
0.6.0	Pavel Kirienko	Nuno Marques	Move the data types to the public_regulated_data_types repository.
0.7.0	Pavel Kirienko	Nuno Marques	Restructure the document, add introduction, define application-level services, add implementation suggestions, add diagrams, remove backlogged content.
0.8.0	Pavel Kirienko	Nuno Marques	Finalize sections "Profiles" and "Topology", add section "Conformance", add examples and elaborations throughout. Move changes to the public document.

Contact and public developer call

For further questions regarding the specification, please contact the maintainers: lorenz@px4.io, nuno@auterion.com and pavel@uavcan.org, or raise issues on the forum. . This standard is being developed by the UAVCAN Drone Standard Special Interest Group, hosted by the Dronecode Foundation.

Trademark guideline

UAVCAN is a registered trademark for the transport and application layer protocols. This standard message set is endorsed by UAVCAN, however, this special interest group and its members do not represent UAVCAN as a protocol design body, but only this message set. This message set is an approved standard of the Dronecode Foundation.

License and disclaimer

Terminology and acronyms

- **UAVCAN** – Uncomplicated Application-layer Vehicular Computing And Networking (standard).
- **RPC** – Remote Procedure Call.
- **Network service** – application-layer functionality whose behavioral contracts are defined in terms of UAVCAN interaction primitives. Not to be confused with *RPC*.
- **UAVCAN subject** – a category of messages exchanged between the participants of a distributed intravehicular computing system pertaining to the same application-layer function or process.
- **Node** – participant of a distributed intravehicular computing system.
- **COTS** – Commercial Off-The-Shelf (equipment).
- **ESC** – Electronic Speed Controller (of an electric motor).

Introduction

This standard defines the standardized application layer for drones and the suitable physical connectivity layer optimized for unmanned vehicles implementing complex behaviors while ensuring a high degree of functional safety. The intermediate network abstraction layers are addressed by the UAVCAN networking technology that this standard is based upon. Therefore, to understand DS-015, one needs to understand what UAVCAN is.

UAVCAN stands for *Uncomplicated Application-layer Vehicular Computing And Networking*. It provides publish-subscribe and RPC (*remote procedure call*) interactions for real-time intravehicular distributed systems, similar to DDS or ROS, but with a reduced capability set to manage the complexity of implementation, verification, and validation.

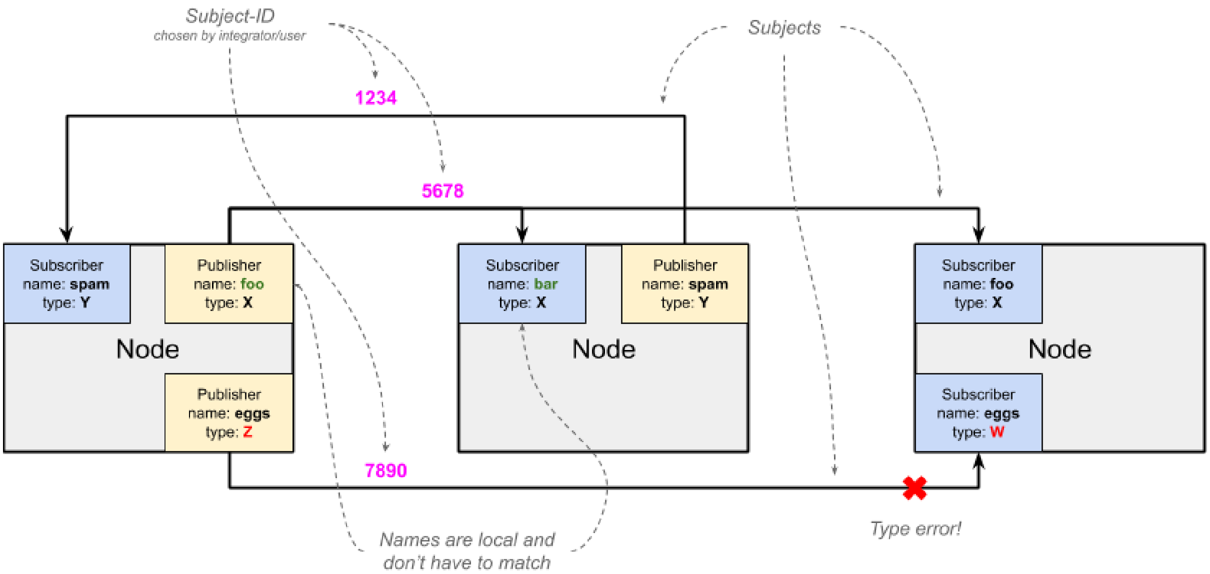
Decentralized anonymous publish-subscribe is the main interaction model where the data links are organized into *subjects*. Subjects are functionally similar to topics found in most other publish-subscribe technologies with the critical difference being that instead of a textual name, subjects are identified by their numeric identifier — a *subject-ID*. The subject-ID is chosen by the network integrator (the engineer responsible for constructing the distributed system), excepting some special circumstances where it is rigidly pre-defined by some other applicable standard (this principle is similar to well-known TCP/UDP port numbers).

A network participant (called *node*) may publish and/or subscribe to an arbitrary number of subjects provided that every node that joins a subject uses a compatible data type for serializing and deserializing data objects exchanged over that subject. The data types are defined using a domain-specific language called DSDL — *Data Structure Description Language*. A large part of this standard is dedicated to defining DSDL types and the related *network services*. DSDL data type definitions are organized into *namespaces* much like classes in an object-oriented programming language.

Despite the fact that subjects are identified by their numeric identifiers rather than names, they may still be named in order to differentiate between different subjects published by or subscribed to by a given node. Since this distinction is confined to the same network participant, the names of the subject bear no relevance for the runtime performance of the network and it is possible that multiple nodes connected to the same subject name it differently. For example, node A may publish the setpoint for a control loop over some subject X, while node B responsible for effecting the control output would subscribe to the same subject; in this example, node A may name the subject **`output_setpoint`** while node B may name it **`command`**. As long as the involved nodes agree on the subject-ID value of X and the data types used for this subject are compatible, they will be able to communicate successfully despite the fact that *internally* they refer to the same subject under different names.

Nodes are configured using the *register API* — an application-layer capability that offers a trivial strongly typed key-value storage per node used to set and read that node's configuration options. The register API is accessed using the RPC functionality mentioned earlier. Fundamentally, RPC is like publish-subscribe with the key difference being that every interaction involves bidirectional data exchange that is not anonymous. The common term covering both subjects and services is *port*; similarly, there is *port-ID*.

The key principles explained above are illustrated in the following diagram. While this introduction is not expected to be an exhaustive description of UAVCAN and is not expected to provide sufficient knowledge for implementing a compliant product, it should be sufficient to serve as a minimal primer. Readers wishing to learn more should turn to **The UAVCAN Guide** and the **UAVCAN Specification**.



Contrary to some prior art, this standard makes heavy emphasis on service-oriented design of its network services as a response to the growing complexity and safety requirements of the addressed vehicular applications. This is covered in chapter **Application layer**.

Even though UAVCAN is capable of supporting different underlying transport layer protocols and it shields the upper protocol layers from the specifics of the underlying transport, this standard is focused primarily on UAVCAN/CAN – a transport defined on top of Classic CAN and CAN FD protocols. The only part of this standard that is not transport-agnostic is the chapter **Physical layer** and the related conventions and recommendations. Future revisions may be extended to cover other transports depending on the demands of the addressed vehicular applications.

The first few chapters are the normative part of this specification that defines the abstract requirements to conformant implementations. To aid understanding, they are followed by various predefined options and examples provided in chapter **Profiles**.

The final chapter **Conformance** addresses issues pertaining to conformance testing.

Protocol layers and standards

Per the OSI model, this standard covers the layers 1 and 7. The OSI layer mapping is provided in the left column for reference.

L7	**UAVCAN Standard Application Layer** (defined in UAVCAN Specification)	**DS-015 Drone Application Layer** (this standard)	Optional vendor-specific application layer services
L5~L6	**Presentation Layer** (defined in UAVCAN Specification)		
L2~L4	**Transport Layer** (defined in UAVCAN Specification)		
L1	**DS-015 Drone Physical Layer** (this standard)		

Application layer

The application layer is defined in two parts:

- The standard application layer is defined by the UAVCAN Specification. Being highly generic, the network services and data types defined there are common for all UAVCAN-capable systems and components regardless of their application domain.

- The Drone Application Layer, being part of this standard, is defined in the regulated namespace `reg.drone` maintained by the UAVCAN project at **github.com/UAVCAN/public_regulated_data_types**.

Additionally, implementers may extend the application layer with custom network services and types. Relevant information can be gathered from **The UAVCAN Guide**.

Presentation layer

The presentation layer is responsible for data representation and fundamental network interaction primitives such as publish-subscribe links and remote procedure calls. This layer is based on the UAVCAN Specification, particularly on DSDL – a domain-specific Data Structure Description Language.

Transport layer

The transport layer is responsible for transferring blocks of binary data between network participants, including its decomposition, reassembly, prioritization, routing, timing management, etc. This layer is based on the UAVCAN Specification.

Physical layer

The physical layer is responsible for electromechanical compatibility of the network participants. It is defined in the Physical layer chapter of this document with references to the applicable third-party standards.

Application layer

UAVCAN standard application layer

UAVCAN defines certain functionality and conventions that are expected to be useful in any vehicular application regardless of its domain. Instead of re-defining its own alternatives, DS-015 relies on such standard functionality offered by UAVCAN. Most of the functions covered in this section are optional per the underlying UAVCAN standard, but mandatory per this DS-015 standard.

Despite the fact that UAVCAN itself implements a flat peer network that is devoid of centralized responsibilities, in the context of the application layer it may be convenient to define a special class of nodes that are responsible for auto-configuring other nodes in the network, barring manual intervention by a human. A typical example of such a node would be the mission computer or a flight management unit. In this document, such nodes are referred to as *autoconfiguration authority* nodes.

Conventions

Design conventions, physical notation conventions, and other conventions specified in section **5.2 Application-level conventions** of the UAVCAN Specification shall be followed.

Conformant implementations are not allowed to use unregulated fixed port identifiers.

Heartbeat

The UAVCAN Specification requires that every node shall publish a heartbeat message at least once per second. This requirement is therefore redundant but is listed here for completeness.

Introspection

The following network services, optional per the UAVCAN Specification, are mandatory to support:

- Node introspection service `uavcan.node.GetInfo`.

Register API

The register API and the standard registers are defined in `uavcan.register.Access`.

The following standard registers shall be supported:

- **`uavcan.node.id`** – 65535 (unconfigured/PnP) by default.
- **`uavcan.node.description`** – empty by default.
- **`uavcan.cookie`** – empty by default.
- **`uavcan.(pub|sub|cln|srv).`*`PORT_NAME`*`.id`** – 65535 (invalid) by default.
- **`uavcan.(pub|sub|cln|srv).`*`PORT_NAME`*`.type`** – constant, set by vendor.

Where ***PORT_NAME*** stands for the name of a publisher, subscriber, server, or client.

A **`uavcan.(pub|sub|cln|srv)`** register set shall be provided for every port for which there is no fixed port-ID. For example, suppose that a node provides electric power estimates of type **`reg.drone.phy.electricity.PowerTs.1.0`** over some subject that is referred to as **`electric_power`** in its documentation. Then, per the requirements, it would have the following registers (among others):

- **`uavcan.pub.electric_power.id`** of type **`natural16`**, persistent and mutable, that the integrator will use to link the subject with the subscribers.

- **`uavcan.pub.electric_power.type`** of type **`string`**, persistent and immutable, that contains a constant string **`reg.drone.phy.electricity.PowerTs.1.0`** defined by the vendor.

Plug-and-play

The plug-and-play (PnP) capability is an optional functionality that allows new nodes to auto-configure their node-ID upon connection to the network, provided that there is a *plug-and-play allocator* available online. Such auto-configured nodes are said to implement the *plug-and-play allocatee* (sic!) behaviors. The DS-015 standard requires that a PnP allocator be always available on the network.

An autoconfiguration authority node would typically implement the PnP allocator logic, being the autoconfiguration authority that grants node-IDs to other network participants. Barring that, the node shall implement the PnP allocatee logic, while also supporting manual assignment of the static port-ID.

A Python implementation available in PyUAVCAN can be consulted with.

Bootloader

Autoconfiguration authority nodes are exempted from these requirements. Other nodes shall implement the standard UAVCAN bootloader logic, which involves supporting the following services:

- uavcan.node.ExecuteCommand (server) — specifically, at least the following commands shall be supported both in the bootloader mode and in the application mode:
  - `COMMAND_RESTART`
  - `COMMAND_BEGIN_SOFTWARE_UPDATE`
- `uavcan.file.Read` (client) — the file read service is used for downloading the firmware image in the bootloader mode.

While the software update is in progress, it is recommended to emit diagnostic messages of type `uavcan.diagnostic.Record` periodically to keep external observers informed about the status of the update process.

Drone application layer

The drone application layer service definitions are the core piece of this standard. They are specified in the DSDL notation in the namespace **`reg.drone`** stored in the official public regulated data type repository maintained by the UAVCAN team ("reg" is short for "regulated"). Please refer to the enclosed documentation for technical details.

Domain-specific services

Please consult with the above-linked **`reg.drone`** namespace for the formal domain-specific service definitions. The following list provides direct links to some of the domain-specific service definitions; if the links have become invalid, please navigate from the root of the repository manually.

- Air data computer
- Smart battery
- ESC
- Servo
- See the repository for further information.

Implementation guidelines

Later revisions of this document will be expanded with links to examples and reference implementations that can be used as a starting point when building a compliant avionic unit.

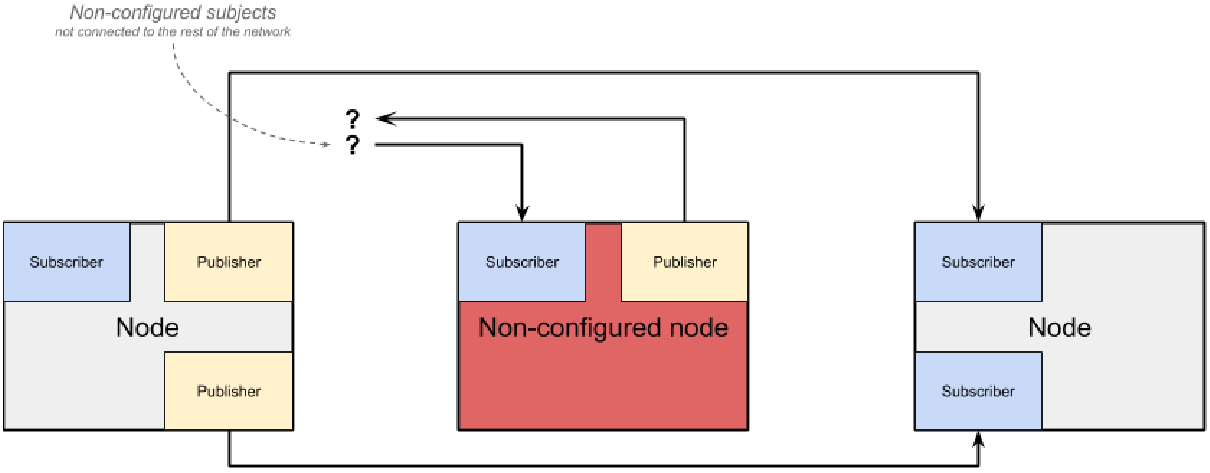
Regardless of the above, the general design process would normally look as follows:

1. Understand the basic principles of UAVCAN by skimming through The UAVCAN Guide.
2. Read the documentation for the `reg.drone` namespace to understand how the Drone application layer is composed.
3. Pick a suitable UAVCAN library (e.g., libcanard) or a full-featured example application to use it as a starting point.
4. Build a minimal UAVCAN node that is only capable of publishing its heartbeat.
5. Gradually add the required standard application layer services listed in the previous section of the document. If the implementation is based on a low-level library like libcanard, this step may require implementing the required application-layer functions like the Register API manually, although this is not expected to require more than a few hundred lines of trivial code.
6. Implement the desired domain-specific services defined in the `reg.drone` namespace.

System integration guidelines

In the absence of advanced autoconfiguration capabilities, a new COTS node is to be integrated into the system following these steps:

1. The new unconfigured node is connected to the network and the vehicle is turned on.
2. The node-ID is configured.
    - If the node-ID has previously been defined statically via the register **`uavcan.node.id`**, the preconfigured static value is used. This is the case if the node is an autoconfiguration authority. Initially, the static node-ID can be configured using some other means of communication like a CLI management interface, MAVLink, etc. Should there happen to be a node-ID conflict on the bus, it is to be resolved by disconnecting one of the conflicting nodes and changing its static node-ID before reconnecting it back.
    - If the node-ID is not defined and the node supports the PnP protocol, it sends PnP node-ID allocation requests until a node-ID is granted by the local autoconfiguration authority.
3. At this point, the node is online but is unable to communicate with the rest of the intravehicular distributed computing system because its subject identifiers are not yet configured. The human integrator sets up the subject-ID parameters using the standard registers defined above to establish the application-layer pub-sub data links with the rest of the network.
4. Once the subject-IDs have been set up, the node is ready to work.



Future capabilities

Future revisions of this standard are expected to provide the following additional capabilities:

- Highly automated plug-and-play node configuration. The intention is to support automatic or semi-automatic node-ID and port-ID configuration for newly connected nodes to reduce the system integration efforts and reduce the risk of misconfiguration by reducing the cognitive workload on the human.
- UAVCAN-MAVLink interoperation services for integration with ground control software.
- Security facilitation services – key exchange, authentication, etc.
- Additional capabilities in the drone application layer (namespace `reg.drone`).

Physical layer

This chapter covers the electromechanical aspects of DS-015: physical layer, connectivity options, electrical and electromechanical design considerations. It contains a set of transport-agnostic recommendations followed by transport-specific sections. This chapter is necessary because the underlying networking standard UAVCAN does not define the physical layer, leaving it to domain-specific standards instead like this one. Following the requirements and recommendations of this chapter will ensure the highest level of inter-vendor compatibility and allow the developers to avoid common design pitfalls.

General recommendations

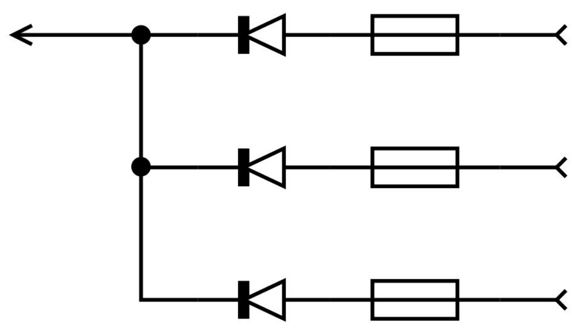
Integrated power supply network

Integration of the power distribution functionality with the communication infrastructure removes the need for a dedicated power distribution network, which has the potential to simplify the system design and reduce the complexity and weight of the wiring harnesses. Redundant power supply topologies can be easily implemented on top of a redundant communication infrastructure.

Power input

A node that draws power from the power supply network should protect its power inputs with an over-current protection circuitry that is capable of disconnecting the input if the power consumption of the node exceeds its design limits. This measure is necessary to prevent a short-circuit or a similar failure of an individual node from affecting other nodes connected to the same power supply network.

In the case of redundant power supply connections where a node is connected to more than one power supply network concurrently, each such connection should be equipped with a circuit that prevents reverse current flow from the node into the power supply network. This measure is necessary to prevent a short-circuit or a similar failure of an individual power supply network from affecting other power supply networks in the same redundant group.

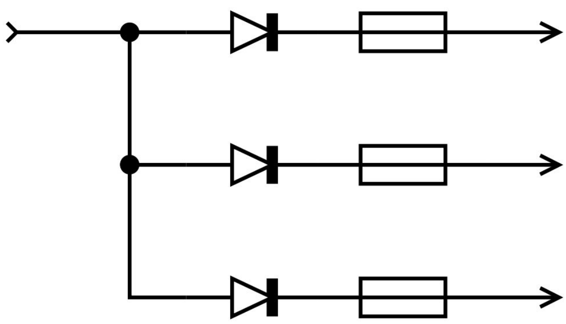


Redundant power input schematic

Power output

A node that delivers power to the power supply network should equip each of its power outputs with a circuit that prevents reverse current flow from the power supply network into the node. This measure is necessary to prevent a short-circuit or a similar failure of the node from affecting the power supply network.

In the case of redundant power output connections where a node provides power to more than one power supply network concurrently, each such connection should be equipped with a circuit that is capable of disconnecting the output if the power consumption per network exceeds the design limits. This measure is necessary to prevent a short-circuit or a similar failure of an individual power supply network from affecting other power supply networks in the same redundant group.



Redundant power output schematic

UAVCAN/CAN recommendations

This section specifies the DS-015 physical layer for UAVCAN/CAN. Here and in the following parts of this section, “CAN” implies both Classic CAN and CAN FD, unless specifically noted otherwise.

Physical connector specification

This standard defines several connector types optimized for different applications: from highly compact systems to large deployments, from low-cost to safety-critical applications.

Each connector type specification includes an integrated power supply interface. Implementations should provide two identical parallel connectors for each CAN interface per device instead of relying on T-connectors.

T-connectors should be avoided because typically they increase the stub length, weight, and complexity of the wiring harnesses.

UAVCAN/CAN Micro connector

The UAVCAN/CAN Micro connector is intended for weight and space-sensitive applications. It is a board-level connector, meaning that it is installed on the PCB rather than on the panel. The Micro connector is compatible with the Pixhawk Connector Standard.

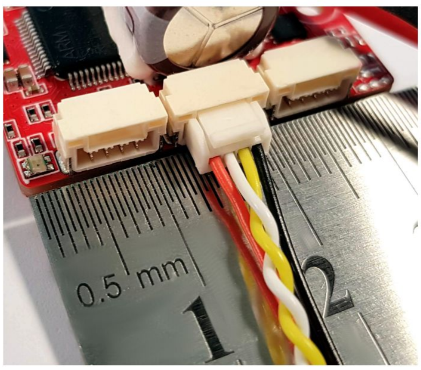
Advantages	Disadvantages
➔ Extremely compact, low-profile. The PCB footprint is under 9×5 millimeters. ➔ Secure positive lock ensures that the connection will not self-disconnect when exposed to vibrations. ➔ Low cost.	➔ Board-level connections only. No panel-mounted options available. ➔ No shielding available. ➔ Not suitable for safety-critical hardware.

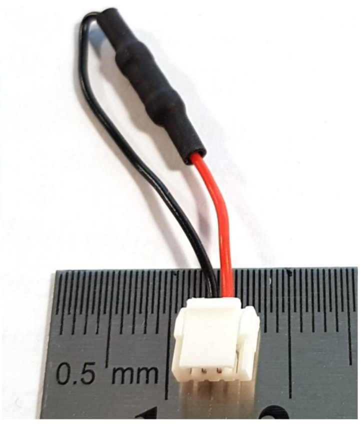
The UAVCAN/CAN Micro connector is based on the proprietary JST GH 4-circuit connector type . The CAN physical layer standard that can be used with this connector type is ISO 11898-2.

Devices that deliver power to the bus are required to provide 4.9–5.5 V on the bus power line, 5.0 V nominal. Devices that are powered from the bus should expect 4.0–5.5 V on the bus power line.

The following table documents the pinout specification for the UAVCAN/CAN Micro connector type. The suitable wire type is #30 to #26 AWG, outer insulation diameter 0.8–1.0 mm, multi-strand. Wires “CAN high” and “CAN low” shall form a twisted pair.

#	Function	Note
1.	Bus power supply	5 V nominal. See the power supply requirements
2	CAN high	Twisted with “CAN low” (pin 3).
3	CAN low	Twisted with “CAN high” (pin 2).
4	Ground	-





UAVCAN/CAN Micro connectors

Right-angle connector with a twisted pair cable connected; a 120Ω termination plug.

Future capabilities

It is recognized that the UAVCAN/CAN Micro connector is unsuitable for many applications, particularly those where ruggedness, reliability, and resilience to adverse environments is required. Future versions of this standard are expected to address this by adding new connector options. The current candidates are listed below for reference; please provide feedback on the forum:

- **UAVCAN/CAN D-Sub**: Generic **D-Subminiature DE-9** with 24V, 3A integrated power. This is the de-facto standard connector for CAN, supported by many current specifications.

- **UAVCAN/CAN M8**: Generic **M8 5-circuit B-coded** with 24V, 3A integrated power. This connector type is also commonly found in various CAN applications and is compatible with the CiA 103 (CANopen) standard.

CAN bus physical layer parameters

Vendors should follow the recommendations provided in this section in the interest of maximizing the cross-vendor compatibility.

Classic CAN

The following table lists the recommended parameters of the ISO 11898-2 Classic CAN 2.0 physical layer. The estimated bus length limits are based on the assumption that the propagation delay does not exceed 5 ns/m, not including additional delay times of CAN transceivers and other components.

Parameter	Value				Unit
Bit rate	1000	500	250	125	kbit/s
Permitted sample point location	75–90	85–90	85–90	85–90	%
Recommended sample point location	87.5	87.5	87.5	87.5	%
Maximum bus length	40	100	250	500	m
Maximum stub length	0.3	0.3	0.3	0.3	m

Designers are encouraged to implement CAN auto bit rate detection when applicable. Refer to the CiA 801 application note for the recommended practices.

The development team [maintains a spreadsheet] that can be used to gauge the Classic CAN network resource utilization.

Note: UAVCAN allows the use of a simple bit time measuring approach, as it is guaranteed that any functioning UAVCAN network will always exchange node status messages, which can be expected to be published at a rate no lower than 1 Hz, and that contain a suitable alternating bit pattern in the CAN ID field. Refer to chapter 5 of the UAVCAN v1 specification for details.

CAN FD

This section is under development and will be populated in a later revision of the standard.

Parameter	Segment	Value				Unit
Bit rate	**Arbitration**	1000	500	250	125	kbit/s
	Data	4000	2000	1000	500	
Permitted SPL	**Arbitration**	TBD	TBD	TBD	TBD	%
	Data	TBD	TBD	TBD	TBD	
Recommended SPL	**Arbitration**	TBD	TBD	TBD	TBD	%
	Data	TBD	TBD	TBD	TBD	
Maximum bus length		TBD	TBD	TBD	TBD	m
Maximum stub length		TBD	TBD	TBD	TBD	

Profiles

CAN bus topologies

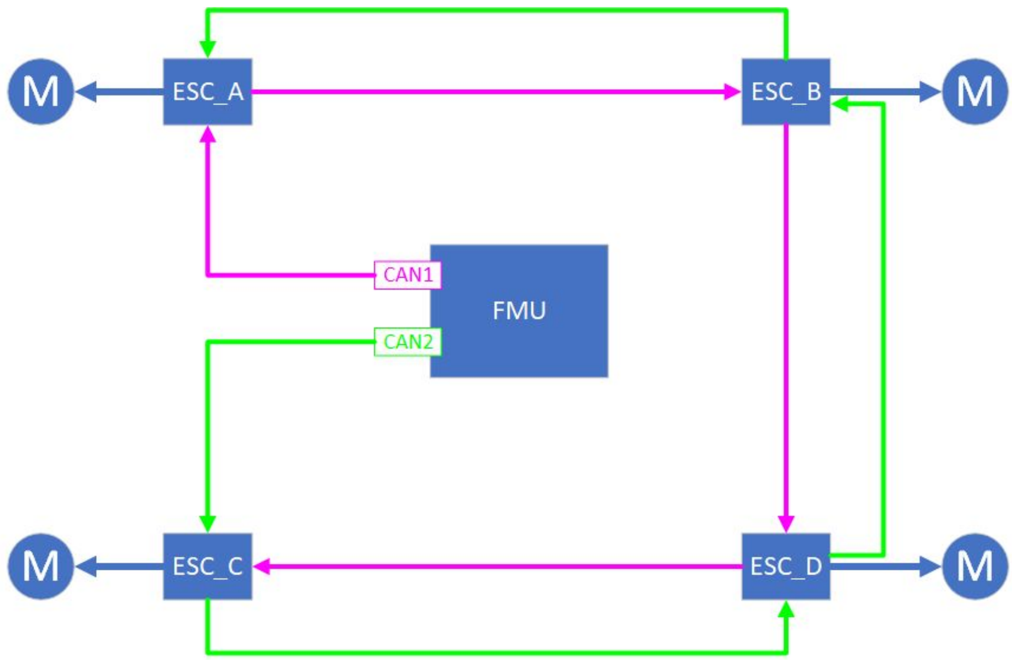
This section lists typical CAN bus topologies recommended for use with this standard. These are not mandatory requirements.

Non-redundant daisy-chain topology

Configurations where a high degree of reliability is not required may implement the conventional topology where a single physical CAN bus is routed through every node in series, forming the typical daisy-chain circuit. This configuration is prone to partitioning should the cable system or at least one of the nodes fail in a mode that disrupts the electrical continuity of the bus.

Redundant daisy-chain topology

In this configuration, the redundant physical buses are routed in the opposite directions to mitigate the partitioning failure mode where one node causes a disruption of the physical buses passing through it, or the redundant cabling system is damaged in the same place.



Quad copter system with dual CAN bus

Star topology

Simple networks where the total cable length is expected to be small or the required data rate is low may leverage the star topology (whether redundant or not) such that the bus is routed through a central passive hub. The hub interconnects all branches into a single electrical network (i.e., it is not an active unit) per redundant bus. Due to signal reflection and the associated signal integrity issues, this topology does not scale to large networks.

Redundant transports

The transport redundancy capability supported by UAVCAN is completely transparent to the application: an application exchanging data over UAVCAN does not have to be aware of the configuration of the underlying transport layer, as the UAVCAN stack automatically fans-out outgoing transfers and deduplicates received ones. It follows that should one of the redundant transports fail at runtime, the UAVCAN stack will automatically find a new configuration that is still functional, shielding the application from the related complexity and transparently handling the edge cases that arise in such scenarios.

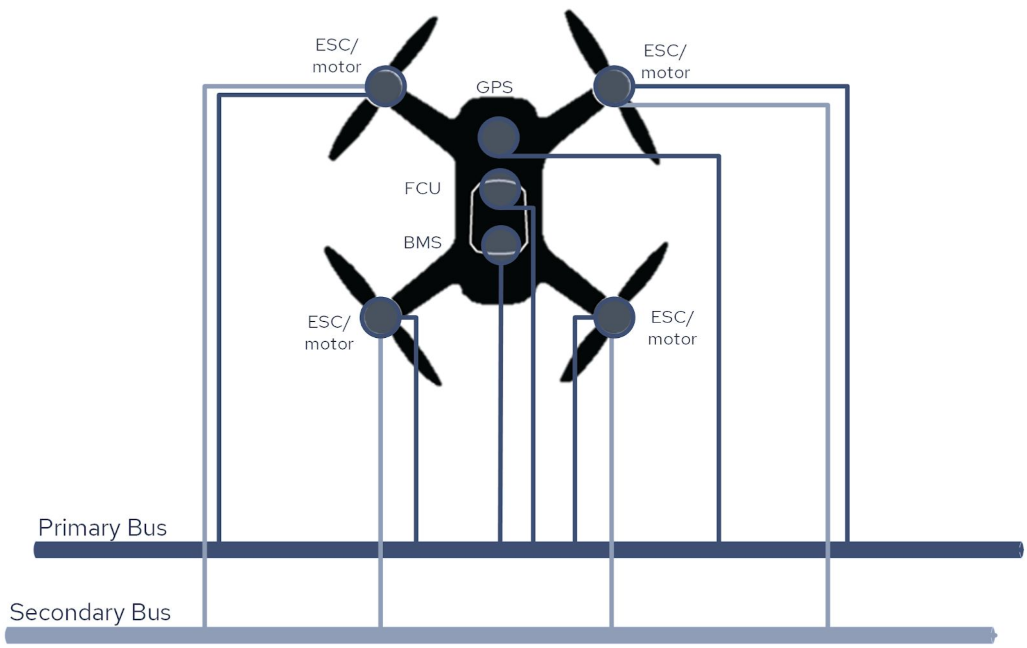
Normally, in a redundant configuration all transports interconnect all of the involved nodes, which is referred to as *symmetric redundancy*. *Asymmetric redundancy* is also supported, where the redundant transports interconnect different sets of nodes:

- The primary transport interconnects all nodes in the network.
- The secondary transport interconnects only the mission-critical nodes in the network, which are the subset of the above.
- The tertiary transport, if present, interconnects yet more critical nodes, which are the subset of the above.

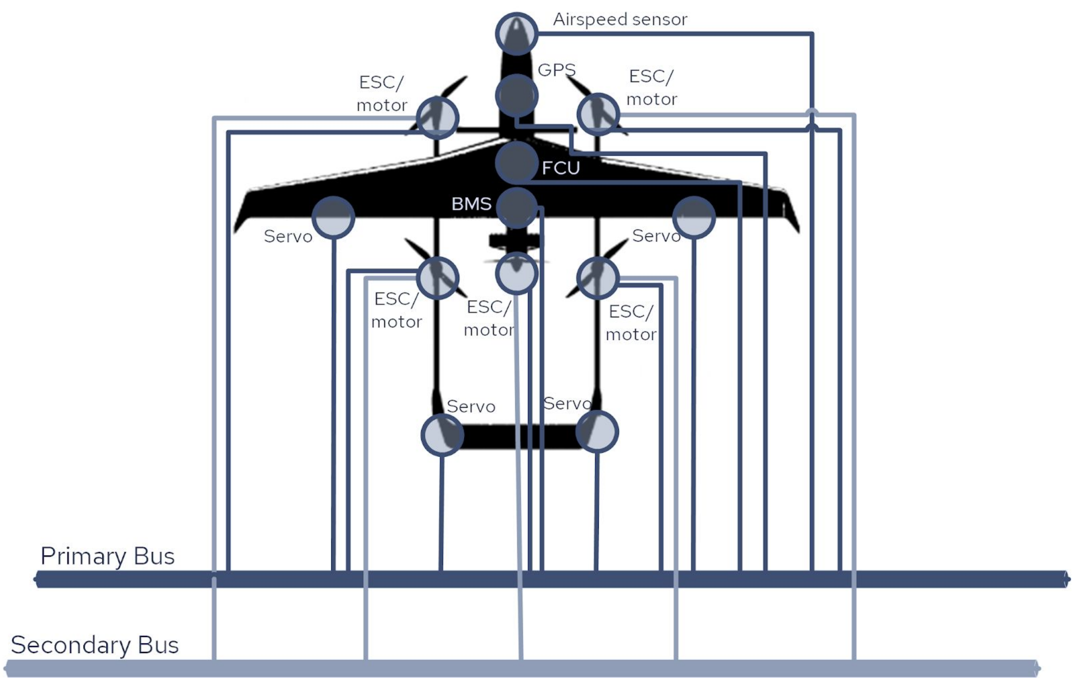
Therefore, nodes that are connected to the tertiary transport use a triply-redundant transport, nodes that are connected to the secondary transport use doubly-redundant transport, and the first category of nodes (non-critical) use the non-redundant transport.

Vehicular topologies examples

Quadrotor



Quad VTOL



Conformance

A detailed description of the conformance testing process is to be defined in a later revision of this standard. A possible course of action is to have a dedicated conformance testing body (or several) that evaluates the submitted products (whether hardware or software) from the standpoint of their compliance to the requirements and contracts set forth by this standard.